Table of Contents

Lustre on Pleiades

Lustre Basics

DRAFT

This article is being reviewed for completeness and technical accuracy.

A Lustre filesystem is a high-performance, shared filesystem (managed with the Lustre software) for Linux clusters. It is highly scalable and can support many thousands of client nodes, petabytes of storage and hundreds of gigabytes per second of I/O throughput.

Main Lustre components:

- Metadata Server (MDS)

  1 or 2 per filesystem; service nodes that manage all metadata operations such as assigning and tracking the names and storage locations of directories and files on the OSTs.
- Metadata Target (MDT)

  1 per filesystem; a storage device where the metadata (name, ownership, permissions and file type) are stored.
- Object Storage Server (OSS)

  1 or multiple per filesystem; service nodes that run the Lustre software stack, provide the actual I/O service and network request handling for the OSTs, and coordinate file locking with the MDS. Each OSS can serve up to ~15 OSTs. The aggregate bandwidth of a Lustre filesystem can approach the sum of bandwidths provided by the OSSes.
- Object Storage Target (OST)

  multiple per filesystem; storage devices where the data in user files are stored. Under Linux 2.6 (current OS on Pleiades), each OST can be up to 8TB in size. Under SLES 11, each OST can be up to 16 GB in size. The capacity of a Lustre filesystem is the sum of the sizes of all OSTs.
- Lustre Clients

  commonly in the thousands per filesystem; compute nodes that mount the Lustre filesystem, and access/use data in the filesystem.

Striping

A user file can be divided into multiple chunks and stored across a subset of the OSTs. The chunks are distributed among the OSTs in a round-robin fashion to ensure load balancing.

Benefits of striping:

- allows one to have a file size larger than the size of an OST
- allows one or more clients to read/write different parts of the same file at the same time and provide higher I/O bandwidth to the file since the bandwidth is aggregated over the multiple OSTs

Drawbacks of striping:

- higher risk of file damage due to hardware malfunction
- increased overhead due to network operations and server contention

There are default stripe configurations for each Lustre filesystem. However, users can set the following stripe parameters for their own directories or files to get optimum I/O performance:

1. stripe_size

   the size of the chunk in bytes; specify with k, m, or g to use units of KB, MB, or GB, respectively; the size must be an even multiple of 65,536 bytes; default is 4MB for all Pleiades Lustre filesystems; one can specify 0 to use the default size.

2. stripe_count

   the number of OSTs to stripe across; default is 1 for most of Pleiades Lustre filesystems (/nobackupp[10-60]); one can specify 0 to use the default count; one can specify -1 to use all OSTs in the filesystem.

3. stripe_offset

   The index of the OST where the first stripe is to be placed; default is -1 which results in random selection; using a non-default value is NOT recommended.

Use the command for setting the stripe parameters:

```
pfe1% lfs setstripe -s stripe_size -c stripe_count -o stripe_offset
dir|filename
```

For example, to create a directory called dir1 with a stripe_size of 4MB and a stripe_count of 8, do

```
pfe1% mkdir dir1
```

```
pfe1% lfs setstripe -s 4m -c 8 dir1
```

Also keep in mind that:

- When a file or directory is created, it will inherit the parent directory's stripe settings.
- The stripe settings of an *existing file* can not be changed. If you want to change the settings of a file, you can create a new file with the desired settings and copy the existing file to the newly created file.

Useful Commands for Lustre

- To list all the OSTs for the filesystem

  ```
  pfe1% lfs osts
  ```

- To list space usage per OST and MDT in human readable format for all Lustre filesystems or for a specific one, for example, /nobackupp10:
  ```
  pfe1% lfs df -h
  pfe1% lfs df -h /nobackupp10
  ```

- To list inode usage for all filesystems or a specific one, for example, /nobackupp10:
  ```
  pfe1% df -i
  pfe1% df -i /nobackupp10
  ```

- To create a new (empty) file or set directory default with specified stripe parameters

  ```
  pfe1% lfs setstripe -s stripe_size -c stripe_count -o
  stripe_offset dir|filename
  ```

- To list the striping information for a given file or directory

  ```
  pfe1% lfs getstripe dir|filename
  ```

- To display disk usage and limits on your /nobackup directory (for example, /nobackupp10):

  ```
  pfe1% lfs quota -u username /nobackupp10
  ```

  or

  ```
  pfe1% lfs quota -u username /nobackup/username
  ```

  To display usage on each OST, add the -v option:

  ```
  pfe1% lfs quota -v -u username /nobackup/username
  ```

Pleiades Lustre Filesystems

Pleiades has several Lustre filesystems (/nobackupp[10-60]) that provide a total of about 3 PB of storage and serve thousands of cores. These filesystems are managed under Lustre software version 1.8.2.

Lustre filesystem configurations are summarized at the end of this article.

Which /nobackup should I use?

Once you are granted an account on Pleiades, you will be assigned to use one of the Lustre filesystems. You can find out which Lustre filesystem you have been assigned to by doing the following:

```
pfe1% ls -l /nobackup/your_username
lrwxrwxrwx 1 root root 19 Feb 23  2010 /nobackup/username -> /nobackupp30/username
```

In the above example, the user is assigned to /nobackupp30 and a symlink is created to point the user's default /nobackup to /nobackupp30.

TIP: Each Pleiades Lustre filesystem is shared among many users. To get good I/O performance for your applications and avoid impeding I/O operations of other users, read the articles: Lustre Basics and Lustre Best Practices.

Default Quota and Policy on /nobackup

Disk space and inodes quotas are enforced on the /nobackup filesystems. The default soft and hard limits for inodes are 75,000 and 100,000, respectively. Those for the disk space are 200GB and 400GB, respectively. To check your disk space and inodes usage and quota on your /nobackup, use the *lfs* command and type the following:

```
%lfs quota -u username /nobackup/username
Disk quotas for user username (uid xxxx):
   Filesystem  kbytes       quota   limit   grace   files   quota   limit   grace
/nobackup/username 1234  210000000 420000000    -     567   75000  100000       -
```

The NAS quota policy states that if you exceed the soft quota, an email will be sent to inform you of your current usage and how much of your grace period remains. It is expected that users will occasionally exceed their soft limit, as needed; however after 14 days, users who are still over their soft limit will have their batch queue access to Pleiades disabled.

If you anticipate having a long-term need for higher quota limits, please send a justification via email to support@nas.nasa.gov. This will be reviewed by the HECC Deputy Project Manager for approval.

For more information, see also, Quota Policy on Disk Space and Files.

NOTE: If you reach the hard limit while your job is running, the job will die prematurely without providing useful messages in the PBS output/error files. A Lustre error with code -122 in the system log file indicates that you are over your quota.

In addition, when a Lustre filesystem is full, jobs writing to it will hang. A Lustre error with code -28 in the system log file indicates that the filesystem is full. The NAS Control Room staff normally will send out emails to the top users of a filesystem asking them to clean up their files.

Important: Backup Policy

As the names suggest, these filesystems are not backed up, so any files that are removed *cannot* be restored. Essential data should be stored on Lou1-3 or onto other more permanent storage.

Configurations

In the table below, /nobackupp[10-60] have been abbreviated as p[10-60].

Pleiades Lustre Configurations

Filesystem	p10	p20	p30	p40	p50	p60
# of MDSes	1	1	1	1	1	1
# of MDTs	1	1	1	1	1	1
size of MDTs	1.1T	1.0T	1.2T	0.6T	0.6T	0.6T
# of usable inodes on MDTs	~235x10^6	~115x10^6	~110x10^6	~57x10^6	~113x10^6	~123x10^6
# of OSSes	8	8	8	8	8	8
# of OSTs	120	60	120	60	60	60
size/OST	7.2T	7.2T	3.5T	3.5T	7.2T	7.2T
Total Space	862T	431T	422T	213T	431T	431T
Default Stripe Size	4M	4M	4M	4M	4M	4M
Default Stripe Count	1	1	1	1	1	1

NOTE: The default stripe count and stripe size were changed on January 13, 2011. For directories created prior to this change, if you did not explictly set the stripe count and/or stripe size, the default values (stripe count 4 and stripe size 1MB) were used. This means that files created prior to January 13, 2011 had those old default values. After this date, directories without an explicit setting of stripe count and/or stripe size adopted the new stripe count of 1 and stripe size of 4MB. However, the old files in that directory will retain their old default values. New files that you create in these directories will adopt the new default values.

Lustre Best Practices

Lustre filesystems are shared among many users and many application processes, which causes contention for various Lustre resources. This article explains how Lustre I/O works, and provides best practices fro improving application performance.

How does Lustre I/O work?

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and object storage targets (OSTs) to perform the I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

Jobs being run on Pleiades content for shared resources in NAS's Lustre filesystem. The Lustre server can only handle about 15,000 remote procedure calls (RPCs, inter-process communications that allow the client to cause a procedure to be executed on the server) per second. Contention slows the performance of your applications and weakens the overall health of the Lustre filesystem. To reduce contention and improve performance, please apply the examples below to your compute jobs, while working in our high-end computing environment.

Best Practices

- **Avoid using *ls -l***

  The *ls -l* command displays information such as ownership, permission and size of all files and directories. The information on ownership and permission metadata is stored on the MDTs. However, the file size metadata is only available from the OSTs. So, the *ls -l* command issues RPCs to the MDS/MDT and OSSes/OSTs for every file/directory to be listed. RPC requests to the OSSes/OSTs are very costly and can take a long time to complete for many files and directories.

  - Use *ls* by itself if you just want to see if a file exists.

  - Use *ls -l filename* if you want the long listing of a specific file.

- **Avoid having a large number of files in a single directory**

Opening a file keeps a lock on the parent directory. When many files in the same directory are to be opened, it creates contention. It is better to split a huge number of files (in the thousands or more) into multiple sub-directories to minimize contention.

- **Avoid accessing small files on Lustre filesystems**

  Accessing small files on the Lustre filesystem is not efficient. If possible, keep them on an NFS-mounted filesystem (such as your home filesystem) or copy them from Lustre to /tmp on each node at the beginning of the job and access them from there.
- **Use a stripe count of 1 for directories with many small files**

  If you have to keep small files on Lustre, be aware that *stat* operations are more efficient if each small file resides in one OST. Create a directory to keep small files, set the stripe count to 1 so that only one OST will be needed for each file. This is useful when you extract source and header files (which are usually very small files) from a tarfile.

  ```
  pfe1% mkdir dir_name
  pfe1% lfs setstripe -s 1m -c 1 dir_name
  pfe1% cd dir_name
  pfe1% tar -xf tarfile
  ```

  If there are large files in the same directory tree, it may be better to allow them to stripe across more than one OST. You can create a new directory with a larger stripe count and copy the larger file to that directory. Note that moving files into that directory with the *mv* command will not change the strip count of the files. Files must be created in or copied to a directory to inherit the stripe count properties of a directory.

  ```
  pfe1% mkdir dir_count_4
  pfe1% lfs setstripe -s 1m -c 4 dir_count_4
  pfe1% cp file_count_1 dir_count_4
  ```

  If you have a directory with many small files (less than 100MB) and a few very large files (greater than 1GB), then it may be better to create a new subdirectory with a larger stripe count. Store just the large files and create symbolic links to the large files using the *symlink* command.

  ```
  pfe1%  mkdir bigstripe
  pfe1%  lfs setstripe -c 16 -s 4m bigstripe
  pfe1%  ln -s bigstripe/large_file  large_file
  ```
- **Use mtar for creating or extracting a tar file**

  A modified gnu tar command, */usr/local/bin/mtar*, is Lustre stripe aware and will create tar files or extract files with appropriately sized stripe counts. Currently, the number of streps is set to the number of gigabytes of the file.

- **Keep copies of your source on the Pleiades home filesystem and/or Lou**

  Be aware that files under /nobackup[p1,p2,p10-p60] are not backed up. Make sure that you have copies of your source codes, makefiles, and any other important files saved on your Pleiades home filesystem or on Lou, the NAS storage system.

- **Avoid accessing executables on Lustre filesystems**

  There have been a few incidents on Pleiades where users' jobs encountered problems while accessing their executables on /nobackup. The main issue is that the Lustre clients can become unmounted temporarily when there is a very high load on the Lustre filesystem. This can cause a bus error when a job tries to bring the next set of instructions from the inaccessible executable into memory.

  Executables run slower when run from the Lustre filesystem. It is best to run executables from your home filesystem on Pleiades. On rare occasions, running executables from the Lustre filesystem can cause executables to be corrupted. Avoid copying new executable over existing executables of the same within the Lustre filesystem. The copy causes a window of time (about 20 minutes) where the executable will not function. Instead, the executable should be accessed from your home filesystem during runtime.
- **Increase the stripe_count for parallel writes to the same file**

  When multiple processes are writing blocks of data to the same file in parallel, I/O performance is better for large files when the stripe_count is set to a larger value. The stripe count sets the number of OSTs the file will be written to. By default, the stripe count is set to 1. While this default setting provides for efficient access of metadata for example to support "ls -l"&emdash;large files should use stripe counts of greater than 1. This will increase the aggregate I/O bandwidth by using multiple OSTs in parallel instead of just one. A rule of thumb is to use a stripe count approximately equal to the number of gigabytes in the file.

  It is also better to make the stripe count be an integral factor of the number of processes performing the write in parallel so that one achieves load balance among the OSTs. For example, set the stripe count to 16 instead of 15 when you have 64 processes performing the writes.

- **Limit the number of processes performing parallel I/O**

  Given that the numbers of OSSes and OSTs on Pleiades are about a hundred or fewer, there will be contention if a huge number of processes of an application are involved in parallel I/O. Instead of allowing all processes to do the I/O, choose just a few processes to do the work. For writes, these few processes should collect the data from other processes before the writes. For reads, these few processes should read the data and then broadcast the data to others.

- **Stripe align I/O requests to minimize contention**

  Stripe aligning means that the processes access files at offsets that correspond to stripe boundaries. This helps to minimize the number of OSTs a process must communicate for each I/O request. It also helps to decrease the probability that multiple processes accessing the same file communicate with the same OST at the same time.

  One way to stripe-align a file is to make the stripe size the same as the amount of data in the write operations of the program.

- **Avoid repetitive *stat* operations**

  Some users have implemented logic in their scripts to test for the existence of certain files. Such tests generate *stat* requests to the Lustre server. When the testing becomes excessive, it creates a significant load on the filesystem. A workaround is to slow down the testing by adding *sleep* in the logic. For example, the following user script tests the existence of the files WAIT and STOP to decide what to do next.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
 end
```

  When neither the WAIT nor STOP file exists, the loop ends up testing for their existence as fast as possible (on the order of 5000 times per second). Adding a *sleep* inside the loop slows down the testing.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
  sleep 15
 end
```

- **Avoid multiple processes opening the same file(s) at the same time**

  On Lustre filesystems, if multiple processes try to open the same file(s), some processes will not able to find the file(s) and the job will fail.

  The source code can be modified to call the sleep function between I/O operations. This will reduce the occcurence of multiple access attempts to the same file from different processes simultaneously.

```
 100  open(unit,file='filename',IOSTAT=ierr)
      if (ierr.ne.0) then
       ...
      call sleep(1)
      go to 100
      endif
```

  When opening a read-only file in Fortran, use ACTION='read' instead of the default ACTION='readwrite'. The former will reduce contention by not locking the file.

```
open(unit,file='filename',ACTION='READ',IOSTAT=ierr)
```

- **Avoid repetitive open/close operations**

  Opening files and closing files incur overhead and repetitive open/close should be avoided.

  If you intend to open the files for read only, make sure to use **ACTION='READ'** in the open statement. If possible, read the files once each and save the results, instead of reading the files repeatedly.

  If you intend to write to a file many times during a run, open the file once at the beginning of the run. When all writes are done, close the file at the end of the run.

Reporting Problems

If you report performance problems with a Lustre filesystem, please be sure to include the time, hostname, PBS job number, name of the filesystem, and the path of the directory or file that you are trying to access.Your report will help us correlated issues with recorded performance data to determine the cause of efficiency problems.

Lustre Filesystem Statistics in PBS Output File

For a PBS job that reads or writes to a Lustre file system, a Lustre filesystem statistics block will appear in the PBS output file, just above the job's PBS Summary block. Information provided in the statistics can be helpful in determining the I/O pattern of the job and assist in identifying possible improvements to your jobs.

The statistics block lists the job's number of Lustre operations and the volume of Lustre I/O used for each file system. The I/O volume is listed in total, and is broken out by I/O operation size.

The following Metadata Operations statistics are listed:

- open/close of files on the Lustre file system
- stat/statfs are query operations invoked by commands such as "ls -l"
- read/write is the total volume of I/O in gigabytes

The following is an example of this listing:

```
=====================================================================
LUSTRE Filesystem Statistics
---------------------------------------------------------------------
  nbp10 Metadata Operations
       open     close      stat    statfs     read(GB)   write(GB)
       1057      1058      1394         0            2          14
Read    4KB    8KB   16KB   32KB   64KB  128KB   256KB   512KB   1024KB
          9      3      1      0      1      0       3       2      319
Write   4KB    8KB   16KB   32KB   64KB  128KB   256KB   512KB   1024KB
        138     13      1     11     36      9      21      37    12479
_____________________________________________________________________
Job Resource Usage Summary for 11111.pbspl1.nas.nasa.gov

    CPU Time Used            : 00:03:56
    Real Memory Used         : 2464kb
    Walltime Used            : 00:04:26
    Exit Status              : 0
```

The read and write operations are further broken down into buckets based on I/O block size. In the example above, the first bucket reveals that nine data reads occurred in blocks between 0 and 4 KB in size, three data reads ocurred with block sizes between 4 KB and 8 KB, and so on. The I/O block size data may be affected by library and system operations and, therefore, could differ from expected values. That is, small reads or writes by the program might be aggregated into larger operations, and large reads or writes might be broken into smaller pieces. If there are high counts in the smaller buckets, you should investigate the I/O pattern of the program for efficiency improvements.

Tips for Improving Lustre I/O

See Lustre Best Practices for multiple tips to improve the Lustre I/O performance of your jobs.